

9. The positioning process (+pruning echo and error values)

$$(x_1 - x_R)^2 + (y_1 - y_R)^2 = d_1^2$$

$$(x_2 - x_R)^2 + (y_2 - y_R)^2 = d_2^2$$

$$(x_3 - x_R)^2 + (y_3 - y_R)^2 = d_3^2$$

suppose: $y_1 \neq y_2$

$$A = \frac{d_1^2 - d_2^2 + x_2^2 - x_1^2}{2(y_2 - y_1)} + \frac{y_2 + y_1}{2}$$

$$B = \frac{x_2 - x_1}{y_2 - y_1}$$

$$y_R = A - Bx_R$$

$$\Rightarrow ax_R^2 + bx_R + c = 0$$

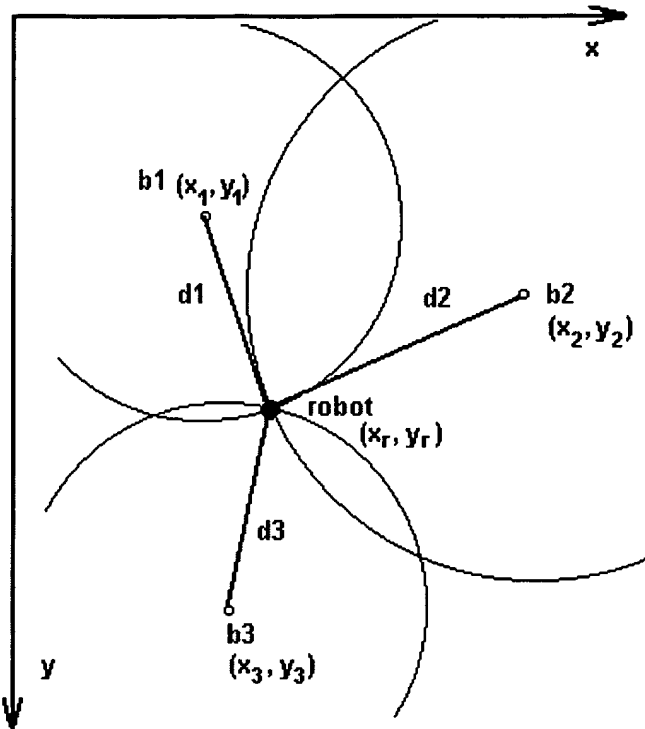
$$a = (1 + B^2)$$

$$b = (2y_1B - 2x_1 - 2AB)$$

$$c = (-d_1^2 + x_1^2 + y_1^2 + A^2 - 2y_1A)$$

if $\Delta = b^2 - 4ac > 0$

$$x_R = \frac{-b \pm \sqrt{\Delta}}{2a}$$



Suppose the beacons coordinates and their respective heights h1, h2, h3 are known; suppose the values r2, r3 are correctly read from the IR-ultrasonic device and transformed to d1, d2, d3; using these equation the 2 solution-points being the intersections of the first two circles should be found easily. The computer must then simply check, which of the solutions fulfills the equation of the remaining circle. If any error occurs, the robot must find its position by dead-reckoning.

The direction and the speed can easily be deduced by comparing former positions to the actual location. The instant position must be concluded from the rotation sensor readings and the motor-actions.

Echo values must be pruned. They occur specially if there are obstacles which deviate the direct wave, or the robot is out of the beam-angle. They are recognizable in most of the cases, because they are much

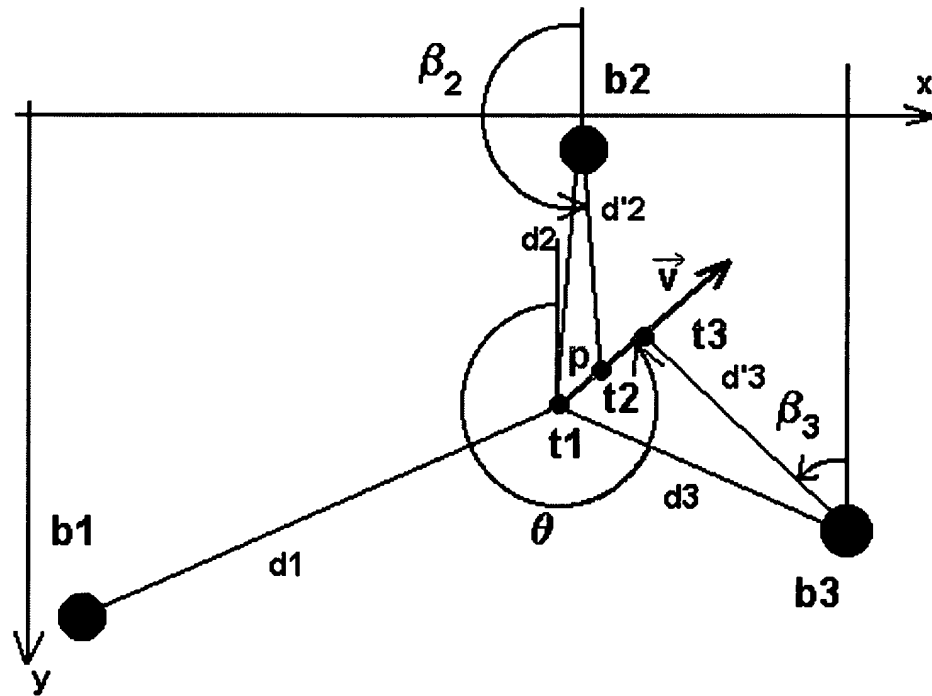
larger than the current values. The robot may change its position only within a certain range depending on its velocity. For example:

$$v = 1 \text{ m/sec}$$

$$\Rightarrow \Delta_{\max} = v \cdot t = 1\text{m, if } t = 1 \text{ sec}$$

so for any d_1 , d_2 , d_3 , the respective increase may not be greater than 1 meter after a travel of 1 sec. This course, has to be adjusted to the robots real velocity.

The beacon3bis.c program shows that the out-of-range error is recognizable by the result 0. A zero distance will never be possible, so this number must be an error. A problem occurs if there is no IR-reception. The receiver simply waits for signal. There is no value-transmission to the robot anymore. In such a case, the beacon3bis.c holds the old value. This situation will only be detectable through dead-reckoning which demonstrate that the actual beacon-deduced position must be erroneous.



Another problem has to be considered. We have already mentioned that the communication protocole between one beacon and the robot needs about 150 ms. If the robot moves at a speed of 1m/sec, its location at beacon2-PING (=t2) will be 15cm from the position at beacon1-PING (=t1). So the positioning algorithm has to take care to find d_2 from d'_2 . Knowing $p = v \cdot (t_2 - t_1) = v \cdot 150\text{ms}$, the angle θ (which is direction of motion) and β_2 (see turnable beacons section), d_2 can be defined through the cosine-triangle-formula. The same should be done for d_3 .

Of course, these calculations may be omitted if the robot's velocity $\leq 0.25 \text{ m/sec}$. In this case the robot will not move more than 4 cm which is the standard-precision of the device.

Obviously we'll need performing trigonometry-approximations in the positioning-task. Here the CORDIC-

functions sine, cosine, tan and arctan to be implanted in the legOS-program. For perfect understanding, have a look at our [CORDIC arctan-page](#). Note that the CORDIC algorithm presents the characteristics to give the sine and cosine **together at the same time**. This might be useful during way-integration, where these values are needed simultaneously. Unlike the arctan-algorithm, the sin/cosine-algorithm decides the sign of k_i by comparing the phase of the rotating vector to the function-argument. After the procedure, the real component corresponds to the cosine, whereas the imaginary component represents the sine. Both values have of course to be divided by the CORDIC-gain.

```

/*The trig-functions sine, cosine, tan, atan, 8-digits precise*/
/*AUTHOR: Claude Baumann 1/5/2001, test-version*/

#include <conio.h>
#include <unistd.h>
#include <dbutton.h>
#include <dkey.h>

#define CORDIC_gain 1.646760258

double Phase (int k) /*in degrees*/
{
    switch(k)
    {
        case 1: return 45.0000000;
        case 2: return 26.5650512;
        case 3: return 14.0362435;
        case 4: return 7.12501635;
        case 5: return 3.57633437;
        case 6: return 1.78991061;
        case 7: return 0.89517371;
        case 8: return 0.44761417;
        case 9: return 0.22381050;
        case 10: return 0.11190568;
        case 11: return 0.05595289;
        case 12: return 0.02797645;
        case 13: return 0.01398823;
        case 14: return 0.00699411;
        case 15: return 0.00349706;
        case 16: return 0.00174853;
        case 17: return 0.00087426;
        case 18: return 0.00043713;
        case 19: return 0.00021857;
        case 20: return 0.00010928;
        case 21: return 0.00005464;
        case 22: return 0.00002732;
        case 23: return 0.00001366;
        case 24: return 0.00000683;
        case 25: return 0.00000341;
        case 26: return 0.00000177;
        case 27: return 0.00000085;
        case 28: return 0.00000042;
        case 29: return 0.00000021;
        case 30: return 0.00000010;
        case 31: return 0.00000005;
        case 32: return 0.00000002;
        case 33: return 0.00000001;
        default: return 888; /*to silence the compiler*/
    }
}

double trig(double alpha,int idx)
{
    double a=1;
    double b=0;
    double kL=1;
    double Theta=0;
    double Temp,beta,tan;
    int L,sign,c;

    beta=alpha;
    c=0;
    while(beta>180) beta-=360; /*beta must be smaller than 180° */
    while(beta<-180) beta+=360; /*beta must be greater than -180° */
    if(beta>90 ) { beta=180-beta; c=1;} /*quadrant II */
    if(beta<-90) { beta=-beta-180; c=1;} /*quadrant III */

```

```

for(L=1;L<34;L++)
{
    if(Theta<beta) { sign=1; } else { sign=-1; } /*compare phase to argument*/
    Theta+=sign*Phase(L);
    Temp=a;
    a-=b*kL*sign;
    b+=Temp*kL*sign;
    kL/=2;
}
a/=CORDIC_gain; /*a=cos(alpha)*/
b/=CORDIC_gain; /*b=sin(alpha)*/
if(c==1) a=-a; /*quadrant matching*/
if(a!=0) {tan=b/a;} else { if(b>0) {tan=999999;} else {tan=-999999;} }
switch(idx)
{
    case 1: return a; /*give the cosine */
    case 2: return b; /* give the sine */
    case 3: return tan; /*give the tan */
    default: return 888; /*to silence the compiler*/
}
}

double sin(double alpha)
{
    return trig(alpha,2);
}

double cos(double alpha)
{
    return trig(alpha,1);
}

double tan(alpha)
{
    return trig(alpha,3);
}

double atan(double x) /* -90 < x < 90 */
{
    double a,b,Temp;
    double kL=1;
    double Theta=90;
    int L,sign;

    if(x>0) {b=1;} else {b=-1;}
    Theta*=-b; /*starting value +90 or -90 depending sign of x*/
    a=x*b; /*a will always be positive */

    Temp=a; /*now rotate by +/- 90°; this could be done more quickly, but for better understanding*/
    a=b*b;
    b=-Temp*b;

    for(L=1;L<34;L++)
    {
        if(b>0) { sign=-1; } else { sign=1; }
        Theta+=sign*Phase(L);
        Temp=a;
        a-=b*kL*sign;
        b+=Temp*kL*sign;
        kL/=2;
    }
    if(x>0) { return Theta+90;} else {return Theta-90;}
}

int main() /*test program*/
{
    int j;
    int p;
    for (j=-10;j<11;j++)
    {
        cputs("x");
        sleep(1);
        lcd_int(j);
        sleep(1);

        cputs("atan");
        sleep(1);
        p=atan(j);
    }
}

```

```

    lcd_int(p);

    while(RELEASED(dbutton()),BUTTON_VIEW)); /*wait until view-button pressed*/
}
return 0;
}

```

Let $b_i(x_i, y_i)$ be the respective coordinates of the beacons and $p_r(x_r, y_r)$ the position of the robot.

$b_i(x_i, y_i)$ are stored in the 4 RCXs as constants when downloading the programs.

Since the ultrasonic-reflector permits horizontal reception, we may suppose the beacons-positions over ground h_i are all equal to zero. cf Preliminary page

$$\Rightarrow d_i = r_i$$

Here now the **TEST** beacon-positioning-procedure, (pruning echo values is still omitted; we use only a slow motion-robot so that speed ≤ 0.25 m/sec):

At the beginning the program must initialize some important values which are needed all over the repeated positionings:

- define the macro SQR(a) ((a)*(b))
- define a constant TOLERANCE ≈ 0
- include a squareroot-approx. SQRT(x)
- include an abs-value function ABS(x)
- include the trig-functions
- Find the two beacons for which the difference $ABS(y_i - y_j)$ is maximized and non-zero. Let's call them b_m and b_n .
- this may be done through the following lines:

```

max:=0;
for i:=1 to 2 do
  for j:=2 to 3 do
    begin
      diff:=abs(y[i]-y[j]);
      if diff>max then begin max:=diff; m:=i; n:=j; k:=6-i-j; end;
    end;

```

Note that the $k:=6-i-j$ is a simple trick:

i	j	k	i+j	6-i-j
1	2	3	3	3

1	3	2	4	2
2	3	1	5	1

- Calculate: $\text{diff} = y_m - y_n$
- Calc: $B = (x_m - x_n) / \text{diff}$
- Calc: $u = (1 + \text{SQR}(B))$

in order to avoid ambiguity, a, b, c of the formulas above are replaced by u, v, w

Note that u is never zero

- repeat forever the fixing of the exact position of the robot:
 - execute a PING-session and store the values r_i
 - Calc: $A = (\text{SQR}(d_n) - \text{SQR}(d_m) + \text{SQR}(x_m) - \text{SQR}(x_n)) / \text{diff} / 2 + (y_m + y_n) / 2$
 - Calc: $v = (2*y_n*B - 2*x_n - 2*A*B)$
 - Calc: $w = (-\text{SQR}(d_n) + \text{SQR}(x_n) + \text{SQR}(y_n) + \text{SQR}(A) - 2*y_n*A)$
 - Calc: $\text{delta} = \text{SQR}(v) - 4*u*w$
 - if $\text{delta} < 0 \implies$ error *Note that the program must respect a certain tolerance due to calculation-error and the 4.5 cm device-tolerance. In these cases let $\text{delta} = 0$*
 - if $\text{delta} = 0 \implies$ 1 solution $x_r = -u / v / 2$
 - if $\text{delta} > 0 \implies$ store the two solutions:
 - $\text{sq_d} = \text{SQRT}(\text{delta})$
 - $x_{r1} = (-v + \text{sq_d}) / u / 2$; $y_{r1} = A - B*x_{r1}$
 - $x_{r2} = (-v - \text{sq_d}) / u / 2$; $y_{r2} = A - B*x_{r2}$
 - Calc: $\text{temp1} = \text{SQR}(x_k - x_{r1}) + \text{SQR}(y_k - y_{r1}) - \text{SQR}(d_k)$ (b_k is the remaining beacon)
 - Calc: $\text{temp2} = \text{SQR}(x_k - x_{r2}) + \text{SQR}(y_k - y_{r2}) - \text{SQR}(d_k)$
 - if $\text{ABS}(\text{temp1}) < (\text{temp2})$, (x_{r1}, y_{r1}) is the solution else it is (x_{r2}, y_{r2})
 - check if $\text{ABS}(\text{temp}_{\text{solution}}) \sim 0 \pm \text{TOLERANCE}$, otherwise error
 - convert the position data to a two-byte message which is sent to the beacons with adding the prefix P
 - the beacons must now calculate the angles $\text{beta}_i = 90 + \arctan((y_i - y_r) / (x_i - x_r))$ and adjust the heading accordingly using the rotation sensor. (consider the cases where $(x_i = x_r)$ The best way to work only with values between -1 and 1 for the arctan-function.)

Here another test program. It searches and displays the values x_{r1} , y_{r1} , x_{r2} and y_{r2} m. It works already very well. Note the changes in the receive-function. **THIS PROGRAM REC_BOT6.c SHOULD BE USED TO TEST THE BEACON-ROBOT COMMUNICATION AND THE POSITIONING.** Put attention to enter the correct beacon-coordinates in the blue marked line.

```
/*rec_bot6.c: should send a message to each beacon in order to PING
Then Calculate and display the values
NEW NEW NEW positioning-calculations*/

#include <unistd.h>
#include <conio.h>
#include <dmotor.h>
```

```

#include <dsensor.h>
#include <string.h>
#include <lnp.h>
#include <lnp-logical.h>

#define aa .0710843 /*linear function to transform raw_1 to real half-byte*/
#define bb -6.9802983
#define cycle 4.3904 /*=1.28E-4 * 343 * 100 result in cm*/

int LOW_read,HIGH_read;

#define POINTS 3
#define SOLUTIONS 2

int x[POINTS]={0,92,0}; /*make these variables global*/
int y[POINTS]={0,75,150}; /*beacon-positions in cm*/
int d[POINTS]={0,0,0};
int i,j,k,m,n,diff,max;
double B,u;

int receive()
{
    int idx,wdt,err;
    long int SUM;

    /*initialize main variables*/
    LOW_read=0;
    HIGH_read=0;
    err=0; /*no error*/
    wdt=0; /*watch-dog-timer*/
    /*communication-protocole*/

    while(SENSOR_1>16000) /*wait until start-message*/
    {
        wdt+=1;
        if (wdt>3000) { err+=1; break; }
    }
    if(err==0) /*continue if no error*/
    {
        wdt=0;
        while(SENSOR_1<16001) /*wait until low_byte*/
        {
            wdt+=1;
            if (wdt>1000) { err+=2; break; }
        }
    }
    /*while waiting until pause-message compute average*/
    if(err==0) /*continue if no error*/
    {
        SUM=0;
        idx=0;
        while(SENSOR_1>16000) /*wait until high_byte*/
        {
            SUM+=SENSOR_1;
            idx+=1;
            if(idx>1000) { err+=3; break; }
        }

        LOW_read=SUM/64/idx; /*gives normal RCX raw-values 0..1023*/
    }

    if(err==0) /*continue if no error*/
    {
        wdt=0;
        while(SENSOR_1<16001)
        {
            wdt+=1;
            if (wdt>1000) { err+=4; break; }
        }
    }
    /*while waiting until stop-message compute average*/

    if(err==0) /*continue if no error*/
    {
        SUM=0;
    }

```

```

idx=0;
while(SENSOR_1>16000)
{
    SUM+=SENSOR_1;
    idx+=1;
    if(idx>1000) { err+=5; break; }
}

HIGH_read=SUM/64/idx; /*gives normal RCX raw-values 0..1023*/
}

if(err==0) /*continue if no error*/
{
    wdt=0;
    while(SENSOR_1<16001) /*wait until NO_SIGNAL condition*/
    {
        wdt+=1;
        if (wdt>1000) { err+=6; break; }
    }
}
return err;
}

int distance(int L, int H)
{
    double raw_1,dist;
    int LOW,HIGH,result;
    /*convert raw sensor-values to half-byte values*/
    raw_1=100000/L;
    raw_1=aa*raw_1+bb;
    LOW=raw_1;
    raw_1=100000/H;
    raw_1=aa*raw_1+bb;
    HIGH=raw_1;

    /*calculate the distance*/
    dist=cycle*(16*HIGH+LOW)+.5;
    result=dist; /*double to integer*/
    return(dist);
}

/*****/

double abs(double x) /*abs value-function*/
{
    if(x>0) {return x;} else { return -x;}
}

double sqr(double x) /*square-function*/
{
    return x*x;
}

double sqrt(double x) /*square-root-function*/
/*Heron's algorithm*/
{
    int idx;
    double q,y;
    q=0;
    y=2;
    for(idx=0;idx<14;idx++) /*this precision is OK, for-loop prevents RCX from
                                getting lost in loop*/
    {
        q=x/y;
        y=(y+q)/2;
    }
    return y;
}

int pos()
{
    double xr[SOLUTIONS];
    double yr[SOLUTIONS];
    int templ,err;
    double A,v,w,sq_d,delta;

```



```

err=0; /*no error yet*/

/*now the calculations*/
A=((sqr(d[n])-sqr(d[m])+sqr(x[m])-sqr(x[n]))/diff+y[m]+y[n])/2;

v=2*(y[n]*B-x[n]-A*B);
w=-sqr(d[n])+sqr(x[n])+sqr(y[n])+sqr(A)-2*y[n]*A;
delta=-4*u*w+sqr(v);
if(delta<0) {err=7; } else
{
    sq_d=sqrt(delta);
    xr[0]=(sq_d-v)/u/2;
    xr[1]=-(v+sq_d)/u/2;
    yr[0]=A-B*xr[0];
    yr[1]=A-B*xr[1];

    templ=xr[0];
    cputs("xr0");
    sleep(1);
    lcd_int(templ);
    sleep(1);

    templ=yr[0];
    cputs("yr0");
    sleep(1);
    lcd_int(templ);
    sleep(1);

    templ=xr[1];
    cputs("xr1");
    sleep(1);
    lcd_int(templ);
    sleep(1);

    templ=yr[1];
    cputs("yr1");
    sleep(1);
    lcd_int(templ);
    sleep(1);
}
return err;
}

/******/

/*CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC*/

int main(int argc, char *argv[])
{
    int L1,L2,L3,H1,H2,H3,E1,E2,E3;
    char *s="X";

    /*positioning computing initialization*/
    max=0;
    for( i=0;i<2;i++) /*find the two beacons which maximize the y[i]-y[j]*/
    {
        for(j=i+1;j<3;j++)
        {
            diff=abs(y[i]-y[j]);
            if(diff>max)
            {
                max=diff;
                m=i;
                n=j;
                k=3-i-j;
            }
        }
    }
}

```

```

diff=y[m]-y[n];
B=(x[m]-x[n])/diff;
u=1+sqr(B);

/*end positioning initialization*/

lnp_logical_range(1); /*set long range*/

/*start the IR / ultrasonic receiver*/
motor_a_dir(fwd);
motor_a_speed(MAX_SPEED);
cputs("wait");
sleep(6);

while(1) /*repeat forever*/
{

    s[0]='A';
    lnp_integrity_write(s,strlen(s)); /*command beacon A to ping*/
    msleep(40); /*wait until PIC no more in error-condition*/
    E1=receive(); /*get the error message*/
    L1=LOW_read; /*get the byte*/
    H1=HIGH_read;

    msleep(25); /*recover-time*/

    s[0]='B';
    lnp_integrity_write(s,strlen(s)); /*command beacon B to ping*/
    msleep(40); /*wait until PIC no more in error-condition*/
    E2=receive(); /*get the error message*/
    L2=LOW_read; /*get the byte*/
    H2=HIGH_read;

    msleep(25); /*recover-time*/

    s[0]='C';
    lnp_integrity_write(s,strlen(s)); /*command beacon C to ping*/
    msleep(40); /*wait until PIC no more in error-condition*/
    E3=receive(); /*get the error message*/
    L3=LOW_read; /*get the byte*/
    H3=HIGH_read;

    lcd_int(E1);
    sleep(1);
    lcd_int(distance(L1,H1));
    sleep(1);

    lcd_int(E2);
    sleep(1);
    lcd_int(distance(L2,H2));
    sleep(1);

    lcd_int(E3);
    sleep(1);
    lcd_int(distance(L3,H3));
    sleep(1);

    d[0]=distance(L1,H1);
    d[1]=distance(L2,H2);
    d[2]=distance(L3,H3);
    if(E1+E2+E3==0) pos(); /*if there is no error, position the robot*/
}
return 0; /*to get rid of compiler warning*/
}

```

Here now the complete robot-program with positioning and sending of message to the beacons in order to adjust their orientations. Note the multi-tasking with a new -still dummy- function called PILOT. This function should be used for navigating the robot. This REC_BOT7.c should be used together with the A_4.c, B_4.c and C_4.c from the turnable beacons section. The correct beacon coordinates must be entered in red marked line. Place the beacons at the correct coordinates, then turn them to direction $\beta=0^\circ$ (sense of the y-axis); start the beacons-programs. Wait the beacons to turn to the initial-positions. Turn on the receiver-task. Move the receiver very slowly around. The displaying slows down the process.

```

/*rec_bot7.c: should send a message to each beacon in order to PING
Then calculate and display the values plus the positioning-calculations
NEW NEW NEW NEW sends the P-message to the beacons and the coordinates
NEW NEW NEW NEW multi-thread: new pilot-task and positioning-task*/

#include <unistd.h>
#include <conio.h>
#include <dmotor.h>
#include <dsensor.h>
#include <string.h>
#include <lnp.h>
#include <lnp-logical.h>

pid_t position_thread;
pid_t pilot_thread;

/*****begin receive routines*****/

#define aa .0710843 /*linear function to transform raw_1 to real half-byte*/
#define bb -6.9802983
#define cycle 4.3904 /*=1.28E-4 * 343 * 100 result in cm*/

int LOW_read,HIGH_read;

int receive()
{
    int idx,wdt,err;
    long int SUM;

    /*initialize main variables*/
    LOW_read=0;
    HIGH_read=0;
    err=0; /*no error*/
    wdt=0; /*watch-dog-timer*/
    /*communication-protocole*/

    while(SENSOR_1>16000) /*wait until start-message*/
    {
        wdt+=1;
        if (wdt>3000) { err+=1; break; }
    }
    if(err==0) /*continue if no error*/
    {
        wdt=0;
        while(SENSOR_1<16001) /*wait until low_byte*/
        {
            wdt+=1;
            if (wdt>1000) { err+=2; break; }
        }
    }
    /*while waiting until pause-message compute average*/
    if(err==0) /*continue if no error*/
    {
        SUM=0;
        idx=0;
        while(SENSOR_1>16000) /*wait until high_byte*/
        {
            SUM+=SENSOR_1;
            idx+=1;
            if(idx>1000) { err+=3; break; }
        }

        LOW_read=SUM/64/idx; /*gives normal RCX raw-values 0..1023*/
    }
}

```

```

    if(err==0) /*continue if no error*/
    {
        wdt=0;
        while(SENSOR_1<16001)
        {
            wdt+=1;
            if (wdt>1000) { err+=4; break; }
        }
    }
    /*while waiting until stop-message compute average*/

    if(err==0) /*continue if no error*/
    {
        SUM=0;
        idx=0;
        while(SENSOR_1>16000)
        {
            SUM+=SENSOR_1;
            idx+=1;
            if(idx>1000) { err+=5; break; }
        }

        HIGH_read=SUM/64/idx; /*gives normal RCX raw-values 0..1023*/
    }

    if(err==0) /*continue if no error*/
    {
        wdt=0;
        while(SENSOR_1<16001) /*wait until NO_SIGNAL condition*/
        {
            wdt+=1;
            if (wdt>1000) { err+=6; break; }
        }
    }
    return err;
}

int distance(int L, int H)
{
    double raw_l,dist;
    int LOW,HIGH,result;
    /*convert raw sensor-values to half-byte values*/
    raw_l=100000/L;
    raw_l=aa*raw_l+bb;
    LOW=raw_l;
    raw_l=100000/H;
    raw_l=aa*raw_l+bb;
    HIGH=raw_l;

    /*calculate the distance*/
    dist=cycle*(16*HIGH+LOW)+.5;
    result=dist;      /*double to integer*/
    return(dist);
}

/*****end receive routines*****/

/*****begin positioning routines*****/

double abs(double x) /*abs value-function*/
{
    if(x>0) {return x;} else { return -x;}
}

double sqr(double x) /*square-function*/
{
    return x*x;
}

double sqrt(double x) /*square-root-function*/
{
    /*Heron's algorithm*/
    int idx;

```

```

double q,y;
q=0;
y=2;
for(idx=0;idx<14;idx++) /*enough precision and prevent
                                program from getting lost in loop*/
{
    q=x/y;
    y=(y+q)/2;
}
return y;
}

#define POINTS 3
#define SOLUTIONS 2
#define TOLERANCE 2500 /*this is the square of the maximum distance of
                        the two-circle intersection solution point to the
                        third circle*/

int x[POINTS]={0,92,0}; /*make these variables global<=====beacon coordinates=====*/
int y[POINTS]={0,75,150}; /*beacon-positions in cm*/
int d[POINTS]={0,0,0};
int i,j,k,m,n,diff,max;
double B,u;
double xr[SOLUTIONS];
double yr[SOLUTIONS];

int pos()
{

    int templ,err;
    double A,v,w,sq_d,delta,temp2,t1,t2;

    err=0; /*no error yet*/

    /*now the calculations*/
    A=((sqr(d[n])-sqr(d[m])+sqr(x[m])-sqr(x[n]))/diff+y[m]+y[n])/2;

    v=2*(y[n]*B-x[n]-A*B);
    w=-sqr(d[n])+sqr(x[n])+sqr(y[n])+sqr(A)-2*y[n]*A;
    delta=-4*u*w+sqr(v);
    if(delta<0) {err=7; } else
    {
        sq_d=sqrt(delta);
        xr[0]=(sq_d-v)/u/2;
        xr[1]=-(v+sq_d)/u/2;
        yr[0]=A-B*xr[0];
        yr[1]=A-B*xr[1];

        templ=xr[0];
        cputs("xr0");
        sleep(1);
        lcd_int(templ);
        sleep(1);

        templ=yr[0];
        cputs("yr0");
        sleep(1);
        lcd_int(templ);
        sleep(1);

        templ=xr[1];
        cputs("xr1");
        sleep(1);
        lcd_int(templ);
        sleep(1);

        templ=yr[1];
        cputs("yr1");
        sleep(1);
        lcd_int(templ);
    }
}

```

```

        sleep(1);

        t1=abs(sqr(x[k]-xr[0])+sqr(y[k]-yr[0])-sqr(d[k])); /*test the third circle*/
        t2=abs(sqr(x[k]-xr[1])+sqr(y[k]-yr[1])-sqr(d[k]));

        if (t2<t1)
        {
            /*exchange the solution, index0 will point the true solution*/
            temp2=xr[0];
            xr[0]=xr[1];
            xr[1]=temp2;

            temp2=yr[0];
            yr[0]=yr[1];
            yr[1]=temp2;

            temp2=t2;
            t2=t1;
            t1=temp2;

        }
        if (t1>TOLERANCE) err=8;

    }
    return err;
}

void P_message()
{
    char *st="Pms";
    int x,y;
    x=xr[0]/6; /*this precision is enough for the beacons to fix the robot*/
    if (x>127) x=127; /*value still signed */
    if (x<-128) x=-128;
    x+=128; /*to make an unsigned char*/
    y=yr[0]/6;
    if (y>127) y=127;
    if (y<-128) y=-128;
    y+=128;
    st[1]=x;
    st[2]=y;
    lnp_integrity_write(st,strlen(st)); /*send generally P-message*/
    msleep(40); /*wait PIC error-condition to recover*/
}

/*****End Positioning routines*****/

/*****Begin Positioning task*****/

int position()
{
    int L1,L2,L3,H1,H2,H3,E1,E2,E3,error;
    char *s="X";

    /*positioning initialization*/
    max=0;
    for( i=0;i<2;i++) /*find the two beacons which maximize the y[i]-y[j]*/
    {
        for(j=i+1;j<3;j++)
        {
            diff=abs(y[i]-y[j]);
            if(diff>max)
            {
                max=diff;
                m=i;
                n=j;
                k=3-i-j;
            }
        }
    }
}

```

```

diff=y[m]-y[n];
B=(x[m]-x[n])/diff;
u=1+sqr(B);

/*end positioning initialization*/

while(1) /*repeat forever*/
{

    s[0]='A';
    lnp_integrity_write(s,strlen(s)); /*command beacon A to ping*/
    msleep(40); /*wait until PIC no more in error-condition*/
    E1=receive(); /*get the error message*/
    L1=LOW_read; /*get the byte*/
    H1=HIGH_read;

    msleep(25); /*recover-time*/

    s[0]='B';
    lnp_integrity_write(s,strlen(s)); /*command beacon B to ping*/
    msleep(40); /*wait until PIC no more in error-condition*/
    E2=receive(); /*get the error message*/
    L2=LOW_read; /*get the byte*/
    H2=HIGH_read;

    msleep(25); /*recover-time*/

    s[0]='C';
    lnp_integrity_write(s,strlen(s)); /*command beacon C to ping*/
    msleep(40); /*wait until PIC no more in error-condition*/
    E3=receive(); /*get the error message*/
    L3=LOW_read; /*get the byte*/
    H3=HIGH_read;

    lcd_int(E1);
    sleep(1);
    lcd_int(distance(L1,H1));
    sleep(1);

    lcd_int(E2);
    sleep(1);
    lcd_int(distance(L2,H2));
    sleep(1);

    lcd_int(E3);
    sleep(1);
    lcd_int(distance(L3,H3));
    sleep(1);

    d[0]=distance(L1,H1);
    d[1]=distance(L2,H2);
    d[2]=distance(L3,H3);
    if(E1+E2+E3==0) /*if there is no error, position the robot*/
    {
        error=pos();
        if (error==0) { P_message(); }
        else
        {
            cputs("err");
            sleep(1);
            lcd_int(error);
            sleep(1);
        }
    }

}

return 0; /*to get rid of the compiler messages*/
}

```

```

/*****End Positioning task*****/

/*****Begin Pilot routines*****/

/*****End Pilot routines*****/

/*****Begin Pilot task*****/
int pilot()
{
    /*do nothing yet but loop*/
    while(1) msleep(10);

    return 0;
}

/*****End Pilot task*****/

/*MAIN MAIN MAIN MAIN MAIN MAIN MAIN MAIN MAIN MAIN MAIN*/

int main(int argc, char *argv[])
{

    lnp_logical_range(1); /*set long range*/

    /*start the IR / ultrasonic receiver*/
    motor_a_dir(fwd);
    motor_a_speed(MAX_SPEED);
    cputs("wait");
    sleep(6);
    position_thread=execi(&position,0,0,PRIO_NORMAL+5,DEFAULT_STACK_SIZE);
    pilot_thread=execi(&pilot,0,0,PRIO_NORMAL,DEFAULT_STACK_SIZE);

    return 0; /*to get rid of compiler warning*/
}

```

Note that error messages AND echo values are pruned. So if such a condition is generated somewhere, there will be no new positioning!

If the robot gets out-of-range errors from one beacon, in a future program, the concerned beacon may be ordered to sweep for finding the robot again. The same might be done if echo-values occurred (recognizable by error-messages 7 or 8!!!). In this case however, all the beacons should sweep, because there is no test yet, which beacon caused the echo value.

HERE NOW THE FIRST REAL-TIME TEST PROGRAM: rec_bot8.c ! Use with A_5.c , B_5.c and C_5.c ! There is no display at all. Proceed as for the previous program. Yeah!!! It works!

```

/*rec_bot8.c: should send a message to each beacon in order to PING
Then calculate the distance-values plus the positioning-calculations
NEW NEW NEW NEW sends the P-message to the beacons and the coordinates
NEW NEW NEW NEW multi-thread: new pilot-task and positioning-task
+ some corrections NEW no display*/

#include <unistd.h>

#include <dmotor.h>
#include <dsensor.h>
#include <string.h>
#include <lnp.h>
#include <lnp-logical.h>

pid_t position_thread;
pid_t pilot_thread;

```



```

/*****begin receive routines*****/

#define aa .0710843 /*linear function to transform raw_1 to real half-byte*/
#define bb -6.9802983
#define cycle 4.3904 /*=1.28E-4 * 343 * 100 result in cm*/

int LOW_read,HIGH_read;

int receive()
{
    int idx,wdt,err;
    long int SUM;

    /*initialize main variables*/
    LOW_read=0;
    HIGH_read=0;
    err=0; /*no error*/
    wdt=0; /*watch-dog-timer*/
    /*communication-protocole*/

    while(SENSOR_1>16000) /*wait until start-message*/
    {
        wdt+=1;
        if (wdt>3000) { err+=1; break; }
    }
    if(err==0) /*continue if no error*/
    {
        wdt=0;
        while(SENSOR_1<16001) /*wait until low_byte*/
        {
            wdt+=1;
            if (wdt>1000) { err+=2; break; }
        }
    }
    /*while waiting until pause-message compute average*/
    if(err==0) /*continue if no error*/
    {
        SUM=0;
        idx=0;
        while(SENSOR_1>16000) /*wait until high_byte*/
        {
            SUM+=SENSOR_1;
            idx+=1;
            if(idx>1000) { err+=3; break; }
        }

        LOW_read=SUM/64/idx; /*gives normal RCX raw-values 0..1023*/
    }

    if(err==0) /*continue if no error*/
    {
        wdt=0;
        while(SENSOR_1<16001)
        {
            wdt+=1;
            if (wdt>1000) { err+=4; break; }
        }
    }
    /*while waiting until stop-message compute average*/

    if(err==0) /*continue if no error*/
    {
        SUM=0;
        idx=0;
        while(SENSOR_1>16000)
        {
            SUM+=SENSOR_1;
            idx+=1;
            if(idx>1000) { err+=5; break; }
        }

        HIGH_read=SUM/64/idx; /*gives normal RCX raw-values 0..1023*/
    }
}

```

```

if(err==0) /*continue if no error*/
{
    wdt=0;
    while(SENSOR_1<16001) /*wait until NO_SIGNAL condition*/
    {
        wdt+=1;
        if (wdt>1000) { err+=6; break; }
    }
}
return err;
}

int distance(int L, int H)
{
    double raw_1,dist;
    int LOW,HIGH,result;
    /*convert raw sensor-values to half-byte values*/
    raw_1=100000/L;
    raw_1=aa*raw_1+bb;
    LOW=raw_1;
    raw_1=100000/H;
    raw_1=aa*raw_1+bb;
    HIGH=raw_1;

    /*calculate the distance*/
    dist=cycle*(16*HIGH+LOW)+.5;
    result=dist; /*double to integer*/
    return(dist);
}

/*****end receive routines*****/

/*****begin positioning routines*****/

double abs(double x) /*abs value-function*/
{
    if(x>0) {return x;} else { return -x;}
}

double sqr(double x) /*square-function*/
{
    return x*x;
}

double sqrt(double x) /*square-root-function*/
/*Heron's algorithm*/
{
    int idx;
    double q,y;
    q=0;
    y=2;
    for(idx=0;idx<14;idx++) /*enough precision and prevent
                                                                    program from getting lost in loop*/
    {
        q=x/y;
        y=(y+q)/2;
    }
    return y;
}

#define POINTS 3
#define SOLUTIONS 2
#define TOLERANCE 2500 /*this is the square of the maximum distance of
                        the two-circle intersection solution point to the
                        third circle*/

int x[POINTS]={0,92,0}; /*make these variables global<=====beacon coordinates=====*/
int y[POINTS]={0,75,150}; /*beacon-positions in cm*/
int d[POINTS]={0,0,0};
int i,j,k,m,n,diff,max;
double B,u;
double xr[SOLUTIONS];
double yr[SOLUTIONS];

int pos()
{

```

```

int temp1,err;
double A,v,w,sq_d,delta,temp2,t1,t2;

    err=0; /*no error yet*/

/*now the calculations*/
A=((sqr(d[n])-sqr(d[m])+sqr(x[m])-sqr(x[n]))/diff+y[m]+y[n])/2;

v=2*(y[n]*B-x[n]-A*B);
w=-sqr(d[n])+sqr(x[n])+sqr(y[n])+sqr(A)-2*y[n]*A;
delta=-4*u*w+sqr(v);
if(delta<0) {err=7; } else
{
    sq_d=sqrt(delta);
    xr[0]=(sq_d-v)/u/2;
    xr[1]=-(v+sq_d)/u/2;
    yr[0]=A-B*xr[0];
    yr[1]=A-B*xr[1];

    t1=abs(sqr(x[k]-xr[0])+sqr(y[k]-yr[0])-sqr(d[k])); /*test the third circle*/
    t2=abs(sqr(x[k]-xr[1])+sqr(y[k]-yr[1])-sqr(d[k]));

    if (t2<t1)
    {
        /*exchange the solution, index0 will point the true solution*/
        temp2=xr[0];
        xr[0]=xr[1];
        xr[1]=temp2;

        temp2=yr[0];
        yr[0]=yr[1];
        yr[1]=temp2;

        temp2=t2;
        t2=t1;
        t1=temp2;

    }
    if (t1>TOLERANCE) err=8;

}
return err;
}

void P_message()
{
    char *st="Pms";
    int x,y;
    x=xr[0]/6; /*this precision is enough for the beacons to fix the robot*/
    if (x>127) x=127; /*value still signed */
    if (x<-128) x=-128;
    x+=128; /*to make an unsigned char*/
    y=yr[0]/6;
    if (y>127) y=127;
    if (y<-128) y=-128;
    y+=128;
    st[1]=x;
    st[2]=y;
    lnp_integrity_write(st,strlen(st)); /*send generally P-message*/
    msleep(40); /*wait PIC error-condition to recover*/
}

/*****End Positioning routines*****/

```

```

/*****Begin Positioning task*****/

int position()
{
    int L1,L2,L3,H1,H2,H3,E1,E2,E3,error;
    char *s="X";

/*positioning initialization*/
    max=0;
    for( i=0;i<2;i++) /*find the two beacons which maximize the y[i]-y[j]*/
    {
        for(j=i+1;j<3;j++)
        {
            diff=abs(y[i]-y[j]);
            if(diff>max)
            {
                max=diff;
                m=i;
                n=j;
                k=3-i-j;
            }
        }

        diff=y[m]-y[n];
        B=(x[m]-x[n])/diff;
        u=1+sqr(B);

/*end positioning initialization*/
    }

while(1) /*repeat forever*/
{
    s[0]='A';
    lnp_integrity_write(s,strlen(s)); /*command beacon A to ping*/
    msleep(40); /*wait until PIC no more in error-condition*/
    E1=receive(); /*get the error message*/
    L1=LOW_read; /*get the byte*/
    H1=HIGH_read;

    msleep(25); /*recover-time*/

    s[0]='B';
    lnp_integrity_write(s,strlen(s)); /*command beacon B to ping*/
    msleep(40); /*wait until PIC no more in error-condition*/
    E2=receive(); /*get the error message*/
    L2=LOW_read; /*get the byte*/
    H2=HIGH_read;

    msleep(25); /*recover-time*/

    s[0]='C';
    lnp_integrity_write(s,strlen(s)); /*command beacon C to ping*/
    msleep(40); /*wait until PIC no more in error-condition*/
    E3=receive(); /*get the error message*/
    L3=LOW_read; /*get the byte*/
    H3=HIGH_read;

    msleep(25); /*recover-time*/

    d[0]=distance(L1,H1);
    d[1]=distance(L2,H2);
    d[2]=distance(L3,H3);
    if(E1+E2+E3==0) /*if there is no error, position the robot*/
    {
        error=pos();
        if (error==0) { P_message(); }
    }
}

```

```

        else
        {
            /*do nothing*/
        }
    }
    msleep(40); /*recover-time*/
}

return 0; /*to get rid of the compiler messages*/
}

/*****End Positioning task*****/

/*****Begin Pilot routines*****/

/*****End Pilot routines*****/

/*****Begin Pilot task*****/
int pilot()
{
    /*do nothing yet but loop*/
    while(1) msleep(10);

    return 0;
}

/*****End Pilot task*****/

/*MAIN MAIN MAIN MAIN MAIN MAIN MAIN MAIN MAIN MAIN MAIN*/

int main(int argc, char *argv[])
{


    lnp_logical_range(1); /*set long range*/

    /*start the IR / ultrasonic receiver*/
    motor_a_dir(fwd);
    motor_a_speed(MAX_SPEED);

    sleep(6);
    position_thread=execi(&position,0,0,PRIO_NORMAL+5,DEFAULT_STACK_SIZE);
    pilot_thread=execi(&pilot,0,0,PRIO_NORMAL,DEFAULT_STACK_SIZE);

    return 0; /*to get rid of compiler warning*/
}

```

 [Main Page](#)